Syntax

1. Objectives & Definitions .............................................................. 2
2. Definitions .................................................................................. 3
3. Lexical Rules .............................................................................. 4
4. BNF: Formal Syntactic rules ..................................................... 6
5. Syntax Diagrams ......................................................................... 9
6. EBNF: Extended BNF .................................................................. 10
7. Example: ...................................................................................... 11
8. BNF Statement Derivation .......................................................... 12
9. Parse Trees .................................................................................. 13
10. Ambiguity ................................................................................... 14
11. Abstract Syntax-Concrete Syntax ........................................... 18
1. Objectives & Definitions

• A programming language is a formal notation for describing algorithm by computer using two components:
  ▪ Syntax: What does a program look like?
  ▪ Semantics: What does a program mean?

• Who must use language definitions?
  ▪ Other language designers
  ▪ Implementers
  ▪ Programmers (the users of the language)

• Syntax:
  ▪ It is a set of rules that formally describe the form or structure of the expressions, statements, and program units.
  ▪ Example:
    o The syntax in programming language define the notation used for arithmetic expressions:
      ▪ Infix notation “5 + 6”
      ▪ Prefix notation “+ 5 6”
    o The syntax define the delimiter of a statement: “;”

• Semantics:
  ▪ It is the meaning of any syntactically valid program written in the language, e.g., expressions, statements, and program units. Note that all syntactically correct programs have valid.
• It is hard to describe the meaning of a language:
  o **Informal description**: describe in English the meaning of a construct.
  o **Formal description**: A mathematical model using formal notation to describe each construct.

• Example:
  o The syntax of a C if statement is:

    If (<expression>) <statement>

    The meaning of this statement form is that the current value of the expression is true, the embedded statement is selected for execution.

  o It would be nice to have a formal description for such construct!!

## 2. Definitions

• **Syntax:**
  o Any language whether natural (such as English) or artificial (such as Java) is a set of words of characters from some alphabet.
  o A sequence of words is called a sentence (in English) or a statement (in a programming language).
  o The syntax rules of a language specify which sentences are in the language.
  o There are two types of syntax rules:
    ▪ Lexical rules
    ▪ Syntactic rules
3. Lexical Rules

- **Objectives:**
  - It helps the programmer know how to write a syntactically correct program.
  - Compiler developers use syntax rules to write syntax analyzer or parser to check the validity of a program.
  - Availability of tools to generate lexical and syntax analyzers: Lex and Yacc.

- **Alphabet:**
  - They specify the set of characters that constitute the alphabet of the language.
  - An alphabet (or vocabulary), V, is a finite non-empty set of symbols.
  - Example:
    - <> and # is valid operator in Pascal and not in C.

- **Words:**
  - The way the characters are combined to form a word.
  - A word over V is a finite string of symbols from V.
  - V * is the set of all the words over V. V * is called the closure of V.
  - Example:
    - Java and java are two different variables in C language.

- **Languages:**
  - A language, L, is any subset of V *.
  - A set of rules for forming the words in a language is called a *grammar*. 
- **Syntactic Elements of a language:**
  - Character set
  - Identifiers
  - Operator symbols (e.g., ==, !=, #)
  - Reserved Keywords (e.g., class, For, procedure)
  - Comments
  - Blanks (should not have a space between += in java)
  - Delimiters and Brackets:
    - “;” marks the end of a statement in C and Java.
    - “begin … end” marks the beginning and the end of a block of code in Ada and Pascal
    - “{ … }” marks the beginning and the end of a block in Java and C.

- Expressions
- Statements

**Example: A C Program**

```c
#include <stdio.h>
#include <ctype.h>

void main()
{
    int i, j;
    printf(" Type a single digit to be squared: ");
    scanf("%d", &i);
    if (!isdigit(i))
    
        j = i * i;
        printf(" %d squared is %d ", i, j);
    
    else
        printf("Your input was not a number!!");
}
```

Delimiters
4. BNF: Formal Syntactic rules

- A little of **history**:
  - Fortran was defined using informal definitions (English Description)
  - Algol 60 was defined by a formal definition (a context free grammar) developed by John Backus.

- **Meta-Language**:
  - It is a language that is used to describe other languages.
  - They may be notational or graphical
  - Example: **Backus-Naur Form (BNF)**

- **BNF Definition**:
  
  - A formal definition defined by John Bachus and Peter Naur to express Algol syntax.

  - A set of nonterminals, terminals, and production rules which define legal sentences in a language.

  - A BNF Rule:

    \[
    \text{<program>} ::= \{<\text{stmt}>*\}
    \]

    reads as

    program is defined as zero or several statements.
Another example: Description of while statement

<while_stmt> ::= while ( <logic_expr> ) <stmt>

- **Terminals:**
  - The primitive tokens of the language ("a", "+", "begin",...)

- **Nonterminals:**
  - Enclosed in "<" and ">", such as <prog>

- **Production rules:**
  - A single nonterminal, followed by ":=", followed by a sequence of terminals and nonterminals.

- **MetaSymbols:**
  - “+”: One of more occurrences of the previous element.
  - “*”: Zero or more occurrences of the previous element.
  - “|”: means “or”

- **Example:**

  - Terminals:
    
    "a" "b" "c" "+" "-" ";" "begin" "end"

  - Nonterminals

    <prog> <stmt_list> <stmt> <var> <exp>
Production rules:

<prog> ::= "begin" <stmt_list> "end"
<stmt_list> ::= <stmt>
<stmt_list> ::= <stmt> ; <stmt_list>
<stmt> ::= <var> ":=" <exp>
<var> ::= "a"
<var> ::= "b"
<var> ::= "c"
<exp> ::= <var> "+" <var>
<exp> ::= <var> "-" <var>
<exp> ::= <var>
5. Syntax **Diagrams**

- A pictorial representation of the syntax of a language.

- They are equivalent to BNF:
  - Put the terminals in circles or ellipses
  - Put the nonterminals in rectangles
  - Connect with lines with arrowheads

- Direct mapping to/from EBNF:

<table>
<thead>
<tr>
<th>Nonterminal</th>
<th>Y</th>
<th>[Diagram]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminal</td>
<td>X</td>
<td>[Diagram]</td>
</tr>
<tr>
<td>Sequence</td>
<td>Y1 Y2</td>
<td>[Diagram]</td>
</tr>
<tr>
<td>Alternation</td>
<td>Y1</td>
<td>Y2</td>
</tr>
<tr>
<td>Optional</td>
<td>[Y]</td>
<td>[Diagram]</td>
</tr>
<tr>
<td>Repetition</td>
<td>{Y}</td>
<td>[Diagram]</td>
</tr>
</tbody>
</table>
Example: Conditional Statement:

BNF:

\[ \langle \text{cond} \rangle ::= \text{if} \ \langle \text{expr} \rangle \ \{ \langle \text{stmt} \rangle^+ \} \ |
\]
\[ \text{if} \ \langle \text{expr} \rangle \ \{ \langle \text{stmt} \rangle^+ \} \ \text{else} \ \{ \langle \text{stmt} \rangle^+ \} \]

6. EBNF: Extended BNF

- EBNF extends BNF syntax to make grammars more readable.

- EBNF Symbols:
  - Replace ::= with →
  - No <> around nonterminals
  - Enclose terminals in single quotes
  - Optional constructs in [ ]
  - Repetitions in { } (or use …)
  - Grouping with ( )

- Sequence:
if → "if " test "then" stmt

• Optional: “[ ]”
  
  if → "if " test "then" stmt ["else" stmt]

• Alternative: “|”

  number → integer | real

• Group: “( )”

  exp → var | ( var "+" var )

• Repetition: “{ }”

  ident_list → ident { "," ident}

7. Example:

• BNF

  \[
  \begin{align*}
  \langle \text{exp} \rangle & \;::=\; \langle \text{exp} \rangle+\langle \text{term} \rangle \mid \langle \text{exp} \rangle-\langle \text{term} \rangle \mid \langle \text{term} \rangle \\
  \langle \text{term} \rangle & \;::=\; \langle \text{term} \rangle\ast\langle \text{factor} \rangle \mid \langle \text{term} \rangle/\langle \text{factor} \rangle \mid \\
  \langle \text{factor} \rangle & \;::=\; (\langle \text{exp} \rangle) \mid \langle \text{identifier} \rangle
  \end{align*}
  \]

• EBNF

  exp → term \{ ( ‘+’ | ‘-’ ) term \}
  term → factor \{ ( ‘*’ | ‘/’ ) factor \}
  factor → ‘(‘ exp ‘)’ | identifier
8. BNF Statement Derivation

- Statements are generated using a sequence of applications of syntactic rules.

- A derivation is the process that allows the generation of a statement.

- Example:

  - A Simple assignment Statements

    assign → id := expr
    id → A | B | C
    expr → id + expr
    | id * expr
    | ( expr )
    | id

  - Derivation: A := B * (A + C)

    assign ⇒ id := expr
    ⇒ A := expr
    ⇒ A := id * expr
    ⇒ A := B * expr
    ⇒ A := B * ( expr )
    ⇒ A := B * ( id + expr )
    ⇒ A := B * ( A + expr )
    ⇒ A := B * ( A + id )
    ⇒ A := B * ( A + C )
9. Parse Trees

- Alternative representation for a derivation
- Example: A parse tree for the previous example
10. **Ambiguity**

- A Syntax definition that generates a statement for which there are two or more distinct parse trees or derivations is said to be **ambiguous**.
- Example: A classical example of the ‘Dangling else’:

```
if (x < 0)
if (y < 0)
y = y - 1;
else
y = 0;
```

- Which if statement has the “else” part?

<table>
<thead>
<tr>
<th>if (x &lt; 0)</th>
<th>if (x &lt; 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (y &lt; 0)</td>
<td>if (y &lt; 0)</td>
</tr>
<tr>
<td>y = y - 1;</td>
<td>y = y - 1;</td>
</tr>
<tr>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td>y = 0;</td>
<td>y = 0;</td>
</tr>
</tbody>
</table>

- Example:

  - Another Simple assignment Statements

```
<assign>   →   <id> := <expr>
{id>       →   A | B | C
<expr>     →   <expr> + <expr>
              | <expr> * <expr>
              | ( <expr> )
              | <id>
```

- What is the difference between this syntax and the previous one?
- Two parse trees for \( A := B + C \times A \)
11. Abstract Syntax-Concrete Syntax

- **Definition**: Two syntax rules have the same abstract syntax if they only differ at the lexical level or concrete syntax.
- **Readability might be affected**: Is it better to use `!=` or `≠`?

- Example: For Loop construct:
  - Ada:
    ```ada
    FOR counter IN lowbound..highbound
    LOOP
        Sequence of statements;
    END LOOP;
    ```
  - C:
    ```c
    For (i=0;i<cond;cond){
        Sequence of statements;
    }
    ```